ECE 569 **Lab 4: Moving the UR3e Robotic Arm** Due 11:59 PM Saturday 22nd, 2025	**Name:** Ashish Prasad Kale

Lab Overview

In this lab, you will combine everything you have learned this semester to make the robotic arm follow your own custom trajectory! First, you will create a custom trajectory for the robot to follow. Then, you will implement a general-purpose forward kinematics (FK) and inverse kinematics (IK) solver. After that, you will use your IK solver to find the joint angles necessary to make the robot arm trace your trajectory, saving to a CSV file. After that, you will verify that the robot arm traces your desired trajectory in RViz. Finally, you will upload your CSV file to Gradescope, so Logan can make the robot arm trace out your desired trajectory. A video of your trajectory will be made available to you by the end of the semester.

Your TA Logan has mounted an LED to the end of the robot arm, which can be turned on and off via an entry in the CSV file. When the LED is turned on and the robot moves, you can perform "light-painting" (Google this cool art form if you haven't seen it before!). Figure 0.1 shows a collection of trajectories captured using a long-exposure shot with a camera from the Fall 2023 class. For Fall 2025, you have the ability to turn off the LED during the motion, which will allow you to get creative with your light paintings.

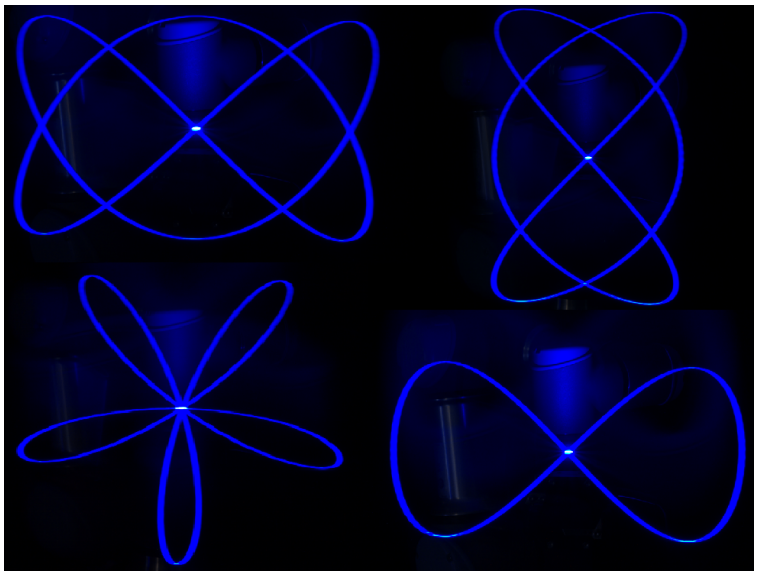

Figure 0.1: Trajectories from Fall 2023

As a word of caution, please start early on this lab. It is by far the most difficult lab to complete. Use Monday office hours to your advantage so that you can get questions answered quickly. Good luck!

Github Link

The Github for Lab 4 is available at https://github.com/purdue-ece569-fall2025/Lab4. You should

Problem 1: Trajectory Generation

Before we can program the robot to move, we need to mathematically define the trajectory that the end-effector of the robot should follow. More specifically, we need to define the homogeneous representation of the end-effector frame $\{b\}$ expressed in the inertial robot base frame $\{s\}$, denoted as $T_{sb}(t)$, at each time $t \in J$ for some interval $J \subseteq \mathbb{R}$.

For our light-painting application, we want the velocity along the majority of the trajectory to be constant, so that the brightness of the trajectory is uniform, as observed by our long-exposure camera. However, given some smooth parametric curve, it is unlikely that the velocity along this curve is constant. We can modify the parametric curve by replacing t with $\alpha(t)$, as shown in the following tutorial.

Creating a Constant Velocity Trajectory

Suppose our desired trajectory is given by a parametric curve with $t \in [0, t_f]$.

$$p_d(t) = \big(x_d(t), y_d(t)\big) \tag{1}$$

The squared velocity of the end effector is given by

$$v^2 = \dot{x}_d^2(t) + \dot{y}_d^2(t)$$

which is (most likely) not constant. Now define

$$x(t) = x_d(\alpha(t)) \tag{2}$$
$$y(t) = y_d(\alpha(t)) \tag{3}$$

where $\alpha(t) : \mathbb{R} \to \mathbb{R}$ is some smooth function which will scale the argument of x_d and y_d. Then, taking derivatives for x and y, we see that the squared velocity is now given by

$$v^2 = \dot{x}^2 + \dot{y}^2 = \dot{\alpha}^2\big(\dot{x}_d^2(\alpha) + \dot{y}_d^2(\alpha)\big) := c^2.$$

where $c > 0$ is the desired constant (average) velocity. (We dropped the explicit dependence on t for brevity.) Solving for $\dot{\alpha}$ yields the differential equation:

$$\dot{\alpha} = \frac{c}{\sqrt{\dot{x}_d^2(\alpha) + \dot{y}_d^2(\alpha)}} := f(\alpha), \qquad \alpha(0) = 0. \tag{4}$$

Finally, we choose the constant (average) velocity c such that the trajectory is completed in t_f seconds with

$$c = \frac{L}{t_f}$$

where L is the arc length of the desired trajectory.

In order to numerically solve for L and α using MATLAB, we can do the following. First, we subdivide time into uniform intervals of length Δt, e.g., $t = [0, \Delta t, 2\Delta t, \ldots, T]$. Then, the arc length is given by

$$L = \sum_{k=1}^{N-1} \sqrt{(x_d[k+1] - x_d[k])^2 + (y_d[k+1] - y_d[k])^2} \tag{5}$$

where N is the length of the time vector. Then $c = L/t_f$. Then we can use the forward-euler method to approximate $\alpha(t)$:

$$\alpha[k+1] = \alpha[k] + f(\alpha[k])\Delta t \tag{6}$$

with $\alpha[1] = 0$, and $f(\alpha)$ is given by (4).

Modifying the Trajectory to have Trapezoidal Velocity Curve

One major issue with a constant velocity trajectory is that at $t = 0$, the robot goes from stationary to moving with velocity c. This discontinuity will make the robot suddenly jolt, which can damage the robot. A simple way to fix this issue is to modify our desired velocity to follow a trapezoidal curve. Figure 0.2 shows the desired velocity profile. The velocity ramps up to its maximum value, remains constant, and then ramps back down to zero. This is ideal for our light-painting application, since the velocity is mostly constant throughout the entire trajectory, and the velocity is zero at terminal conditions $t = 0$ and $t = t_f$.

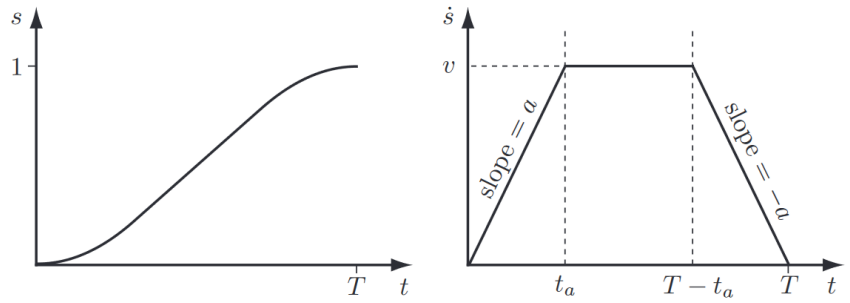

Figure 9.5: Plots of $s(t)$ and $\dot{s}(t)$ for a trapezoidal motion profile.

Figure 0.2: Trapezoidal motion profile, MR Fig. 9.5.

We can modify (4) to ensure that the trapezoidal velocity profile is met. Let $g : \mathbb{R} \to \mathbb{R}$ be the trapezoidal curve given by Figure 0.2, multiplied by an appropriate scalar so that $\frac{1}{t_f}\int_0^{t_f} g(t)dt = 1$. (This normalization ensures that the average velocity is still c, so the entire trajectory is completed in exactly t_f seconds.) Then, we can redefine $\dot{\alpha}$ to be:

$$\dot{\alpha} = \frac{cg}{\sqrt{\dot{x}_d^2(\alpha) + \dot{y}_d^2(\alpha)}} := f(\alpha), \qquad \alpha(0) = 0. \tag{7}$$

(In this step, we just substituted $v(t) = cg(t)$.) Solving with the Forward-Euler method and (7) will result in a trajectory with a trapezoidal velocity profile. This is the method you will use for Task 1. Note that the function g is provided to you in the MATLAB starter code, which takes the arguments t, T, and t_a and returns the height of the trapezoid, $g(t)$.

Task 1

For this step, you will need to do the following.

(a) Choose your own Lissajous curve of the form

$$x_d(t) = A\sin(at)$$
$$y_d(t) = B\sin(bt)$$

replacing the constants A, B, a, b with positive constants such that a/b is rational. You can play around with different Lissajous curves using the this Desmos calculator.

- **(Workspace)** Trajectory must satisfy $|x(t)| \leq 0.16$ and $|y(t)| \leq 0.16$.
- **(Zero Terminal Points)** Trajectory must start and end at $(x, y) = (0, 0)$.
- **(Speed limit)** Robot must complete trajectory within 15 seconds without exceeding speed of $c = 0.25$.

Take a screenshot of your desired trajectory either in Desmos or plotted using MATLAB.

(b) Calculate the arc length of your trajectory numerically, and then find c, the target average velocity. (Note: if the arc length is too long, you will need to choose a different trajectory.) Then, use Forward-Euler method with (7) to numerically approximate $\alpha(t)$. Use a step size of $\Delta t = 1/500$ seconds.

Plot $\alpha[k]$ and take a screenshot. Please insert a dashed line at $t = t_f$ as shown in the example solution. You should ensure that $\alpha(t_f) = T$.

(c) Now, your trajectory is given by

$$x[k] = A\sin(a\alpha[k])$$
$$y[k] = B\sin(b\alpha[k])$$

and the speed of your trajectory is therefore

$$v[k] = \sqrt{\left(\frac{x[k] - x[k-1]}{\Delta t}\right)^2 + \left(\frac{y[k] - y[k-1]}{\Delta t}\right)^2}.$$

Plot $v[k]$ and take a screenshot. Insert a dashed red line at $y = 0.25$, and a dashed black line at $y = c$, as shown in the example solution. The velocity profile should be trapezoidal (it is OK to have minor fluctuations in speed). Include proper units.

Problem 2: Forward Kinematics

Given a list of joint angles, we want to represent the position and orientation of the end effector in the $\{s\}$ frame. For this, we will use the Product of Exponentials using screw axes in the base frame. Using the convention of Section 4.1.1 of the MR book, we denote the base frame as $\{s\}$ and the end-effector frame as $\{b\}$.

Let $M \in SE(3)$ denote the configuration of the end-effector frame in the relative to the base frame, let $S = (S_1, \ldots, S_6) \in \mathbb{R}^{6\times 6}$ denote the screw axes of the robot in the home position in the base frame (and $B = (B_1, \ldots, B_6)$ denote the screw axes of the robot in the end-effector frame), and let $\theta(t) = (\theta_1, \ldots, \theta_6)^T \in \mathbb{R}^6$ denote the joint angles in radians. Then, we can obtain the matrix T_{sb} with either of the following expressions:

$$T_{sb}(t) = e^{\widehat{S_1}\theta_1(t)} \cdots e^{\widehat{S_n}\theta_n(t)} M = M e^{\widehat{B_1}\theta_1(t)} \cdots e^{\widehat{B_n}\theta_n(t)} \tag{8}$$

which represents the end effector frame expressed in the base frame at time t. Recall we can convert $S_i \leftrightarrow B_i$ using the isomorphism:

$$S_i = Ad_M B_i$$

for each $i = 1, \ldots, n$.

(a) Before you implement the forward kinematics function, you will need to implement the following 5 functions. (Note that all functions names are prefixed with ECE569_.)

1. You must implement the **VecTose3** (hat operator) function $f : \mathbb{R}^6 \to se(3)$ defined by the map:

$$\xi\theta \mapsto \widehat{\xi}\theta$$

2. You must implement the **se3ToVec** (vee operator) function $f : se(3) \to \mathbb{R}^6$ defined by the map:

$$\widehat{\xi}\theta \mapsto \xi\theta$$

3. You must implement the **MatrixLog6** function $f : SE(3) \to se(3)$ defined by the map:

$$e^{\widehat{\xi}\theta} \mapsto \widehat{\xi}\theta$$

Note: You may use the provided **MatrixLog3** in the starter code. Be sure to consider the case when $\widehat{\omega}\theta = 0$ separately.

4. You must implement the **MatrixExp6** function $f : se(3) \to SE(3)$ defined by the map:

$$\widehat{\xi}\theta \mapsto e^{\widehat{\xi}\theta}$$

Note: You may use the provided **AxisAng3** and **MatrixExp3** in the starter code. Be sure to consider the case when $\widehat{\omega}\theta = 0$ separately.

5. You must implement the **Adjoint** function $Ad_T : SE(3) \to \mathbb{R}^{6\times 6}$ defined by the map:

$$T \mapsto \begin{bmatrix} R & 0 \\ \widehat{p}R & R \end{bmatrix}$$

MATLAB starter code is available on github https://github.com/purdue-ece569-fall2025/Lab4. Make sure that you add the **funtions** and **tests** folders to your path (see Fig. 0.3).

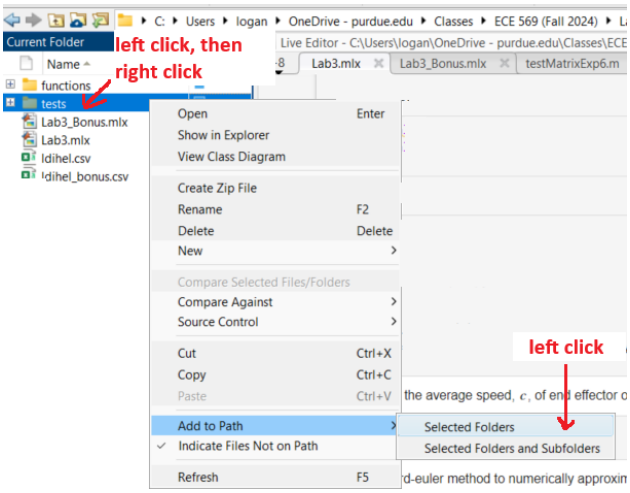

Figure 0.3: Adding folders to path. Be sure to add both the **functions** and **tests** to your path!

Once you have implemented these functions, run their test functions to verify your functions are working as expected. (In MATLAB, run the "runtests2a" script from the Command Window. In Python, run the "runtests2a" script.) **Take a screenshot of the test results.**

(b) Now, implement the forward kinematics functions **FKinSpace** and **FKinBody**. These functions take in the home configuration M, the screw axes (S for spatial, or B for body screw axes, respectively), and the list of joint values θ, and return the homogeneous representation of the end-effector in the spatial frame using (8). Both functions should return the same value provided θ. Even though our ur3e robot has 6 joints, your functions should work for any n-link robot, provided the screw axes and joints are specified. (Hint: how should you loop over the n joints?)

Once you have implemented these functions, run its test bench to verify your functions are working as expected. (In MATLAB, run the "runtests2b" script from the Command Window. In Python, run the "runtests2b" script.) Take a screenshot of the test results.

(c) Now, using M, S, $\theta|_{t=0}$, and , find the matrix $T_0 = T_{sb}(0)$ using your **FKinSpace** and **FKinBody** functions. They should be identical. **Take a screenshot of these matrices, along with their difference.** The matrices M and S and B, along with the vector $\theta|_{t=0}$ are provided in the starter code. For completeness, they are listed here, too.

$$M = \begin{bmatrix} -1.0000 & 0 & 0 & 0.4567 \\ 0 & 0 & 1.0000 & 0.2232 \\ 0 & 1.0000 & 0 & 0.0665 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0000 & 1.0000 & 1.0000 & 0 & 1.0000 \\ 1.0000 & 0 & 0 & 0 & -1.0000 & 0 \\ 0 & -0.1519 & -0.1519 & -0.1519 & -0.1311 & -0.0665 \\ 0 & 0 & 0 & 0 & 0.4567 & 0 \\ 0 & 0 & 0.2435 & 0.4567 & 0 & 0.4567 \end{bmatrix}, \quad \theta|_{t=0} = \begin{bmatrix} -1.6800 \\ -1.4018 \\ -1.8127 \\ -2.9937 \\ -0.8857 \\ -0.0696 \end{bmatrix}$$

(d) For each time step, we can calculate the desired position and orientation of the end-effector expressed in the base frame with

$$T_{sb}(t) = T_0 T_d(t)$$

where

$$T_d(t) = \begin{bmatrix} R_d(t) & p_d(t) \\ 0 & 1 \end{bmatrix}$$

and $p_d(t) = (x(t), y(t), 0)^T$. (You can choose $R_d = I_3$ for simplicity, but the choice is ultimately yours.) The matrix $T_{sb}(t)$ can be decomposed into $(R(t), p(t))$ as follows.

$$T_{sb}(t) = \begin{bmatrix} R(t) & p(t) \\ 0 & 1 \end{bmatrix} \tag{9}$$

Plot $p(t)$ using the plot3 function in MATLAB. Include all appropriate units. Denote the start of the trajectory with a green circle, and the end of the trajectory with a red x.

Problem 3: Inverse Kinematics

Now that we have $T_{sb}(t)$ for each time t, we can use the inverse kinematics algorithm to find the joint angles $\theta(t)$.

1. First, you must implement the following two functions. **When you have finished implementing them, run their test benches and take a screenshot of the results. (In MATLAB, run the "runtests3a" script from the Command Window. In Python, run the "runtests3a" script.)**

   (a) You must implement the **TransInv** function $f : SE(3) \to SE(3)$ defined by the map

   $$T \mapsto T^{-1} = \begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix}$$

   (b) You must implement the **JacobianBody** function, which takes the n screw axes B_i (at the home configuration, represented in the end-effector frame) and joint angles θ and returns the body Jacobian. That is, the function maps

   $$(B, \theta) \mapsto \mathbb{R}^{6\times n}$$

   Recall that the last column of J_b is given by $J_{bn} = B_n$, while the ith column ($i = 1, \ldots, n-1$) is given by

   $$J_{bi}(\theta) = Ad_{T_i}(B_i), \quad T_i = e^{-\widehat{B}_n \theta_n} \cdots e^{-\widehat{B}_{i+1} \theta_{i+1}}$$

2. Implement the **IKinBody** function as described in the MR book, pages 230-231 (or equivalently, the inverese kinematics function from lecture). This function takes in the following:

   - B: The joint screw axes in the end-effector frame when the manipulator is at the home position, in the format of a matrix with the screw axes as the columns,
   - M: The home configuration of the end-effector,
   - T: The desired end-effector configuration T_{sd},
   - θ: An initial guess of joint angles that are close to satisfying T (often the result from the previous iteration)
   - e_ω: A small positive tolerance on the end-effector orientation error. The returned joint angles must give an end-effector orientation error less than e_ω,
   - e_v: A small positive tolerance on the end-effector linear position error. The returned joint angles must give an end-effector position error less than e_v.

   The function should return two values:

- θ^*: Joint angles that achieve T within the specified tolerances,
- success: A logical value where TRUE means that the function found a solution and FALSE means that it ran through the set number of maximum iterations without finding a solution within the tolerances e_ω and e_v. When you have finished implementing this function, **take a screenshot of the test bench "runtests3b".**

3. Now, for each time step $t_2, \ldots, t_N$, run the inverse kinematic function. Your initial guess should be the previous θ^*. After running the inverse kinematics for all time, you should plot the first order difference in joint angles:

$$\Delta\theta[k] = \theta[k+1] - \theta[k]$$

   You need to verify that the first order difference plots are small in magnitude (in the order of 10^{-3}). (Otherwise, try changing the tolerances e_ω and e_v or the number of iterations in your inverse kinematics.) **Take a screenshot of the first order difference in joint angles.**

4. We can verify that our inverse kinematics function worked by running the forward kinematics function on the calculated joint angles. That is, for each time step, use the **FKinSpace** or **FKinBody** function to calculate $T_{sd} = (R(t), p(t))$. Then, create a 3D plot of $p(t)$ just like you did before, changing the title to **Verified Trajectory in {s} frame**. **Take a screenshot of the verified trajectory plot.**

5. Check that your robot does not enter a kinematic singularity by plotting the determinant of the body Jacobian. If this plot touches zero, there is a kinematic singularity. You will need to

6. Use the provided code to save to CSV. The order CSV file should have no header, the column order should be time, joint1, joint2, joint3, joint4, joint5, joint6, and led. The led column should be either 0 or 1, corresponding to whether the led should be off or on at the provided time step. Please save the CSV file as your Purdue username. For example, if your email is **ldihel@purdue.edu** then save the file as **ldihel.csv**.

Problem 4: Verification in RViz

You will need to use RViz to visualize the movement of the robotic arm, ensuring that your prescribed motion is continuous and draws your desired curve. Please see GitHub for instructions for this step. When you are done, take a screenshot of the robot arm and your trajectory in RViz.

In Gradescope, there will be an additional assignment created for submitting your Lab 4 CSV file. Please submit the CSV file for your Lissajous trajectory there. (You may resubmit this CSV file as many times as you'd like. We will arrange a time for in person students to see their trajectories running on the robot!)

Problem 5: Bonus

Create your own trajectory for light-painting. You can turn off the LED by writing a 0 in right-most column of the CSV file for the appropriate time slots. Please be creative! The best trajectories will be shown at the end of the semester to students, and may also be used for next year's course. Your trajectory is subject to the following conditions:

1. **(No Collisions)** the robot must not collide with itself, the table, or the wall. You should use RViz to visually check that there are no collisions.

2. **(Continuous Velocity)** the robot's velocity should be continuous.

3. **(Velocity Limits)** the maximum velocity of the end effector should be reasonable (more than 0.5 m/s may not be possible.)

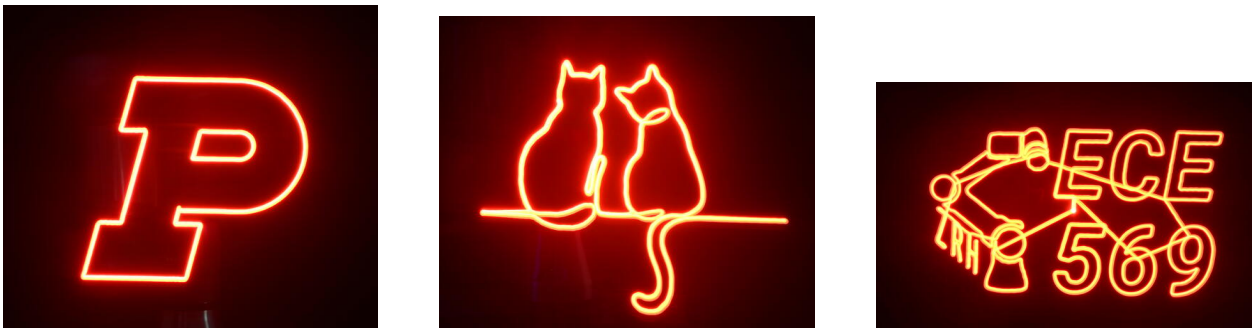

Figure 0.4: Select bonus trajectories from Fall 2024.

4. **(Sample Rate)** the time step should be set to 2 ms, as you did with the Lissajous trajectory.
5. **(Not a spiral)** Too many spirals were submitted in Fall 2024. Push yourself to do something a little more difficult! A smiley face with spiral eyes would be acceptable.
6. **(Must run on robot for full credit)** Your trajectory needs to work on the actual ur3e robotic arm to recieve the maximum bonus points. Check that the trajectory looks good in RViz (the `Show Trace` can help you view the trajectory, and you have plots to check that the robot does not try to go into a singularity) and that the maximum speed of the end effector and joint angles are reasonable (no more than 0.5 m/s and no more than 100 deg/s, respectively). Note that you might not be able to view the entire trajectory in RViz at the same time using the `Show Trace` button, but you might be able to write your own custom ROS node that positions markers at the `tool0` frame at points where the LED is turned on, if you are looking for a deeper understanding of RViz (not required).
7. **(Rated E for everyone)** Please just make sure that all trajectories are appropriate to show to the class.

Here are some ideas for you to get started.

(a) Paint an interesting 3D trajectory, such as the edges of a cube.

(b) Write your initials or name using the LED. (This would look really good on LinkedIn!)

(c) Draw an emoji.

(d) Draw an animal. (Last year people drew lots of cats!)

(e) Write the current time using the LED. You can make this even more impressive by making a function which generates trajectories for every possible time `12:00 AM` to `11:59 PM`.

When you are done and have verified the accuracy of your trajectory in RViz, please submit the CSV file to the Lab 4 CSV Bonus assignment in Gradescope. **You should save this CSV file as your Purdue ID followed by _bonus.** For example, if your email is **ldihel@purdue.edu** then you should save the file as **ldihel_bonus.csv**.

Your TA Logan wrote some notes on minimum jerk trajectories and creating piecewise trajectories for the robot arm, available here if you are interested. This could be helpful for making piece-wise trajectories for the bonus assignment, but is not necessary.

Figures

All of your figures will be placed at the end of the file. This makes grading much faster!

1 Step 1 Figures

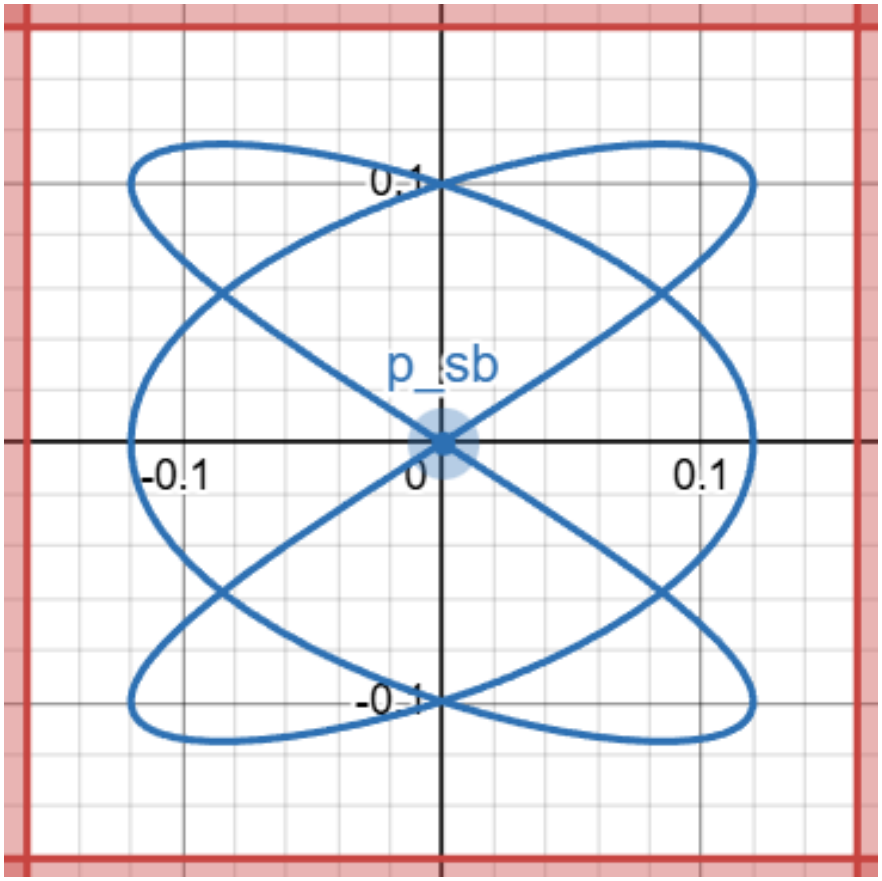

Figure 1.1: Desired Trajectory

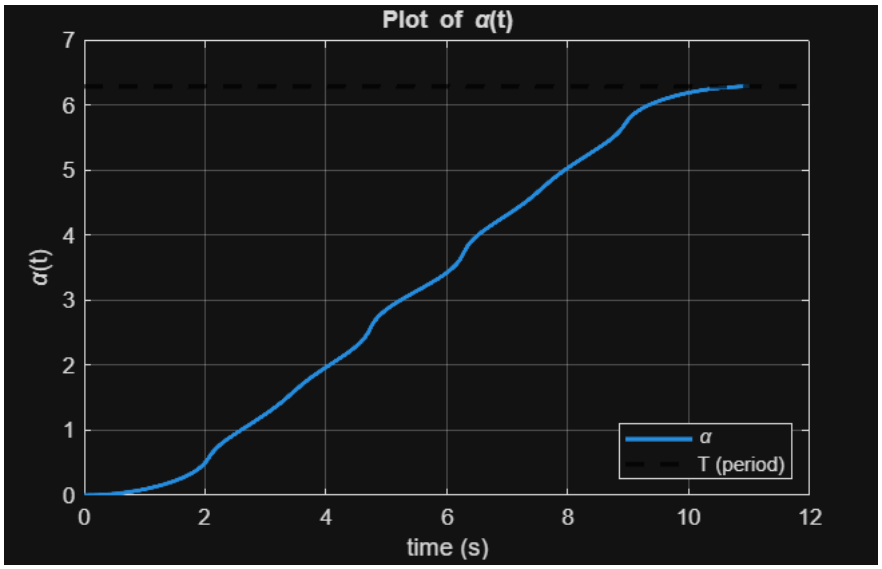

Figure 1.2: Plot of $\alpha[k]$

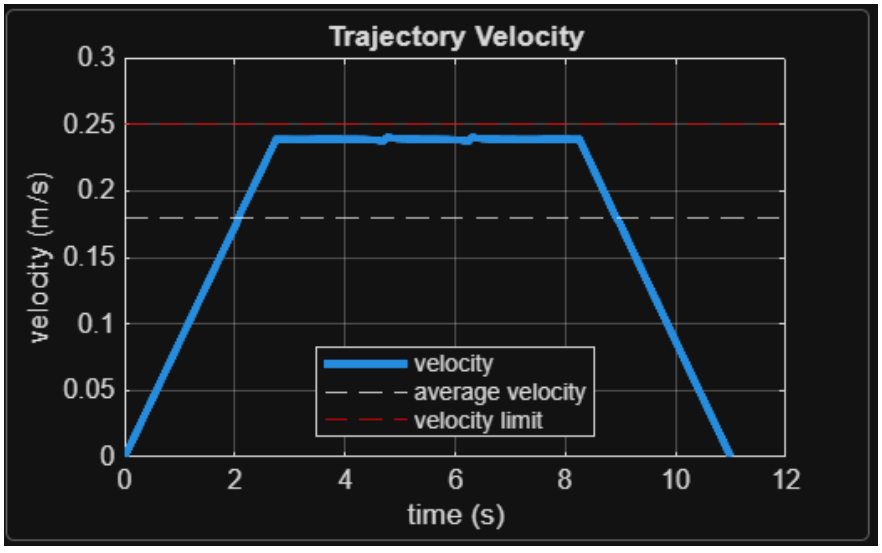

Figure 1.3: Plot of $v[k]$

2 Step 2 Figures

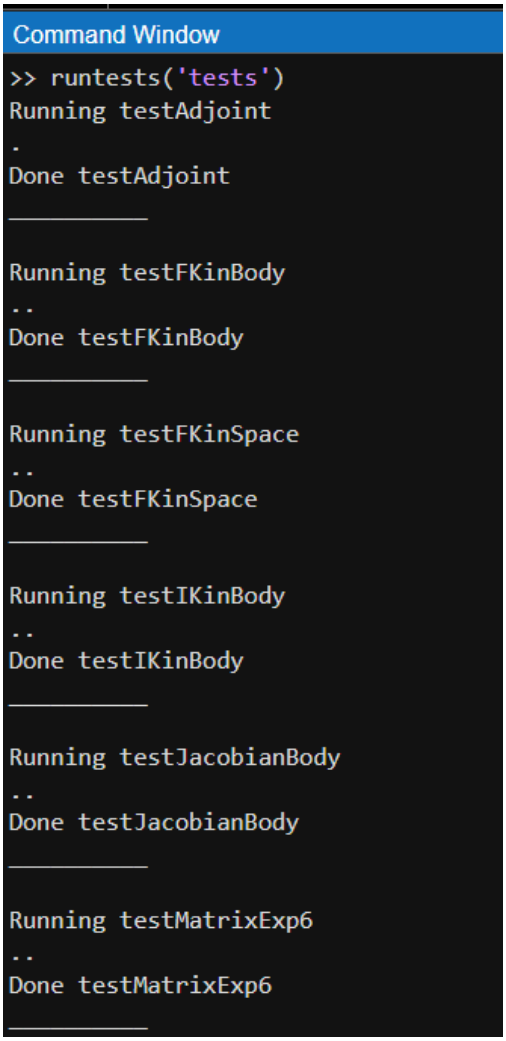

Figure 2.1: Test Bench Results for all tests

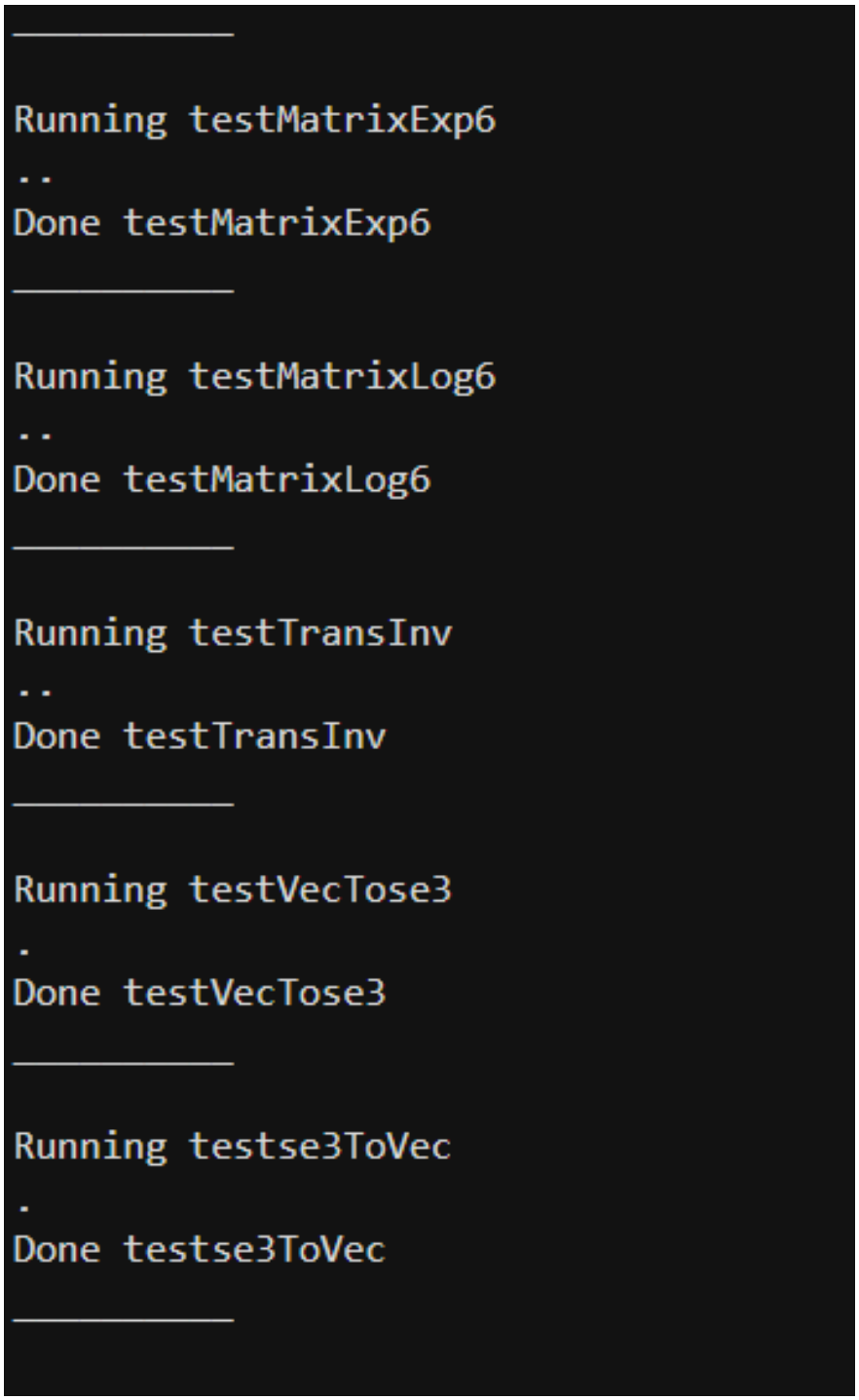

Figure 2.2: Test Bench Results for all tests

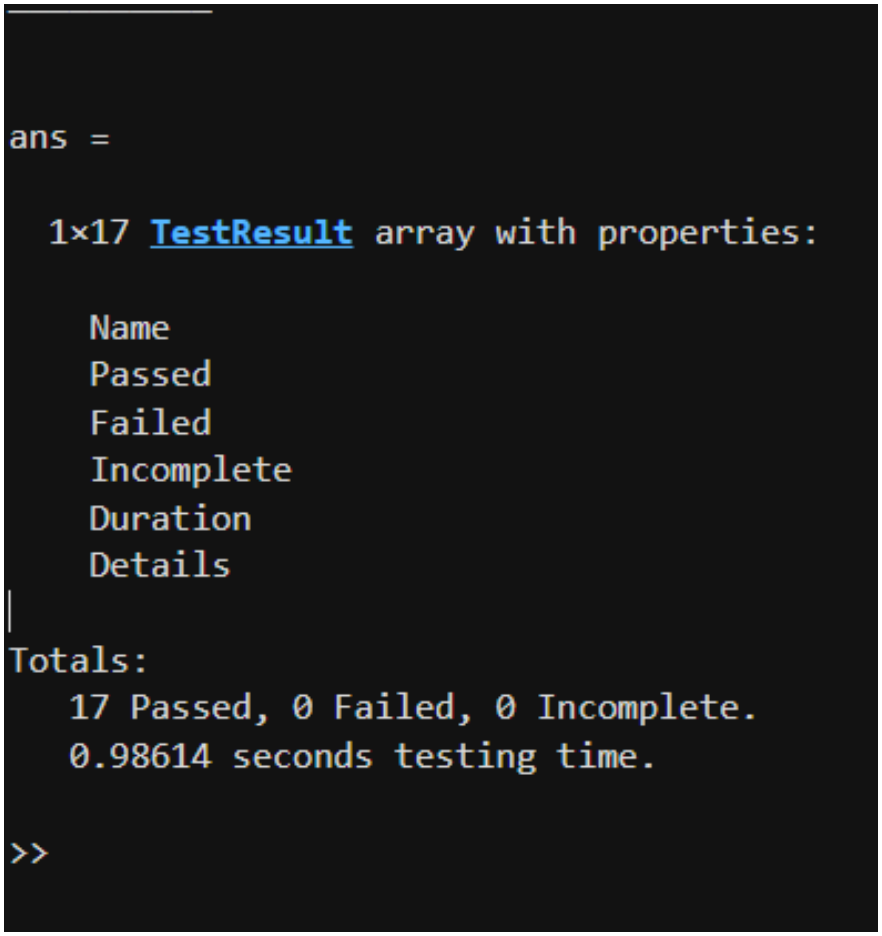

Figure 2.3: Test Bench Results for all tests

```
T0_space = 4×4
    -0.6987   -0.0569    0.7131    0.2154
     0.7151   -0.0248    0.6986    0.2271
    -0.0221    0.9981    0.0580    0.2966
         0         0         0    1.0000

T0_body = 4×4
    -0.6987   -0.0569    0.7131    0.2154
     0.7151   -0.0248    0.6986    0.2271
    -0.0221    0.9981    0.0580    0.2966
         0         0         0    1.0000

ans = 4×4
10^-15 ×
         0   -0.0069         0    0.0278
    -0.1110   -0.0208   -0.1110   -0.1665
    -0.0208    0.1110   -0.0208   -0.1110
         0         0         0         0
```

Figure 2.4: Showing the Equivalence of FKinSpace and FKinBody (or screenshot from python)

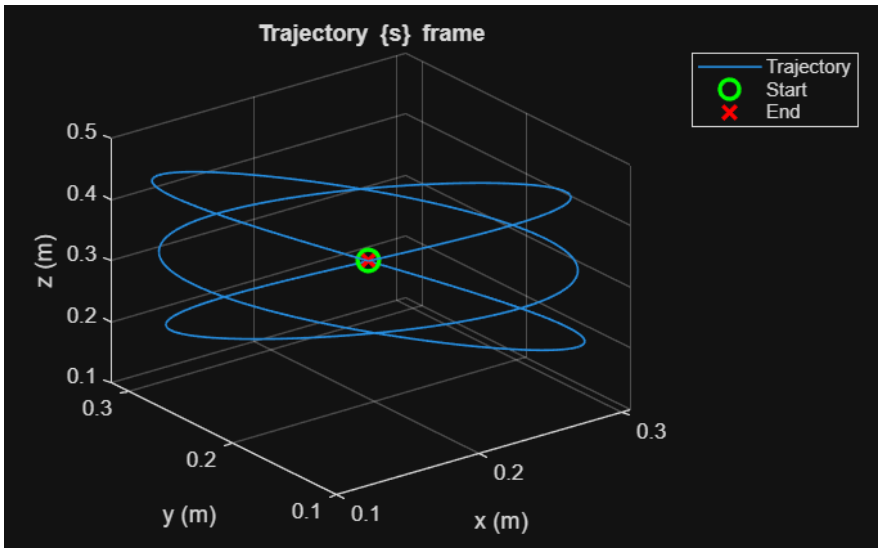

Figure 2.5: Trajectory in the s frame

Figure 2.6: First Order Difference in Joint Angles. (Note the scale of the figure.)

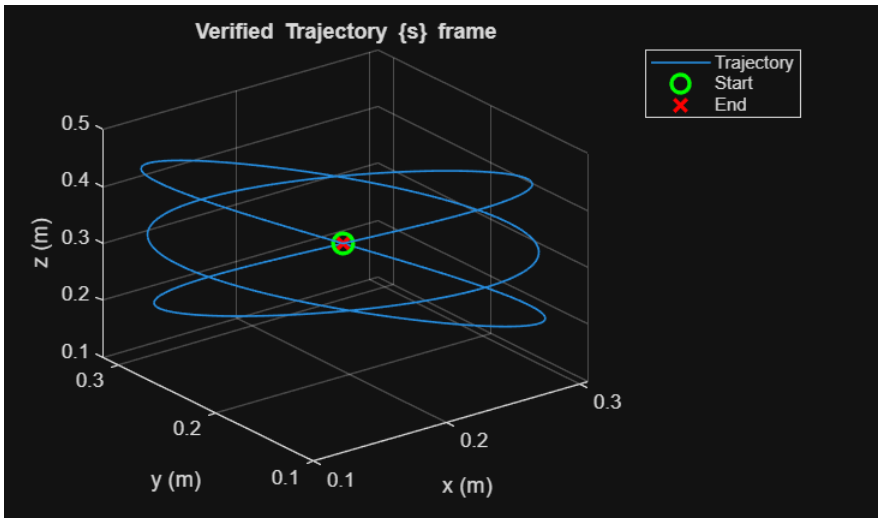

Figure 2.7: Verified Trajectory (using forward kinematics on the output of our inverse kinematics function)

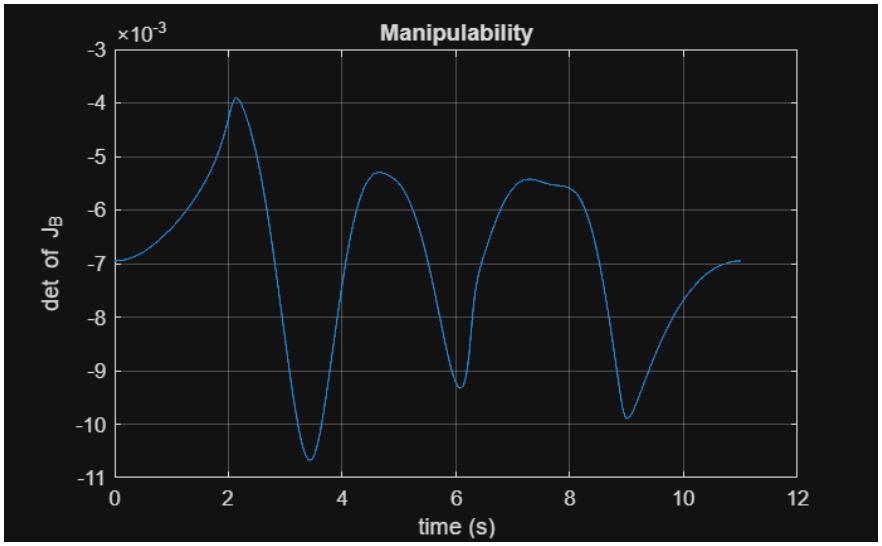

Figure 2.8: Plot of Determinant of Body Jacobian. The plot should stay away from zero, indicating that there are no kinematic singularities.

3 Step 4 Figures

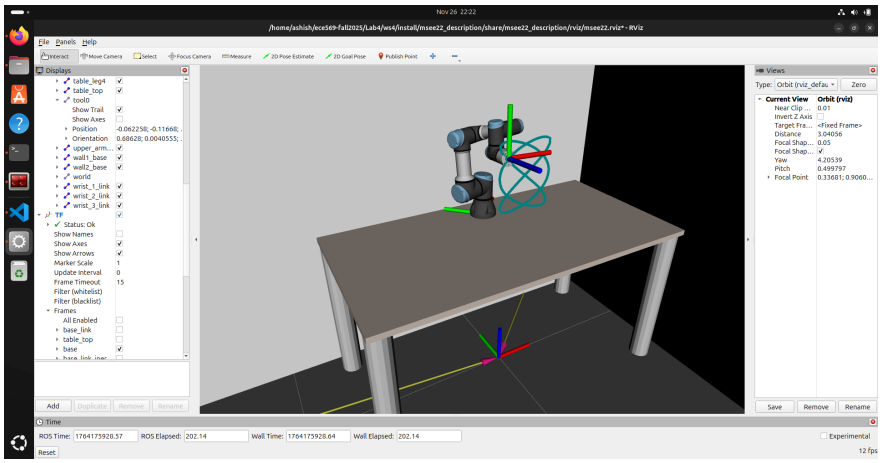

Figure 3.1: Trajectory in RViz

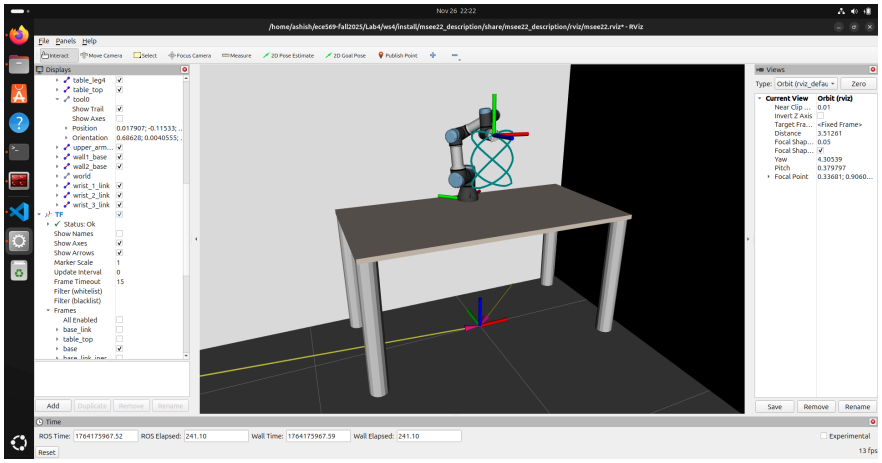

Figure 3.2: Trajectory in RViz

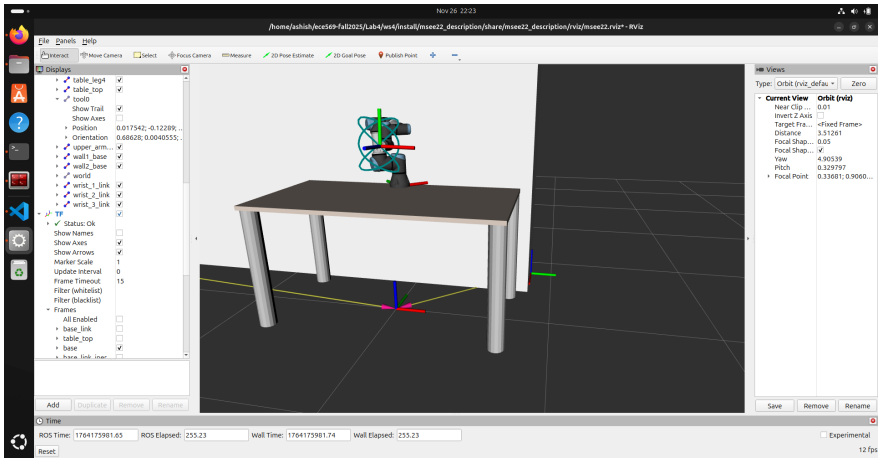

Figure 3.3: Trajectory in RViz

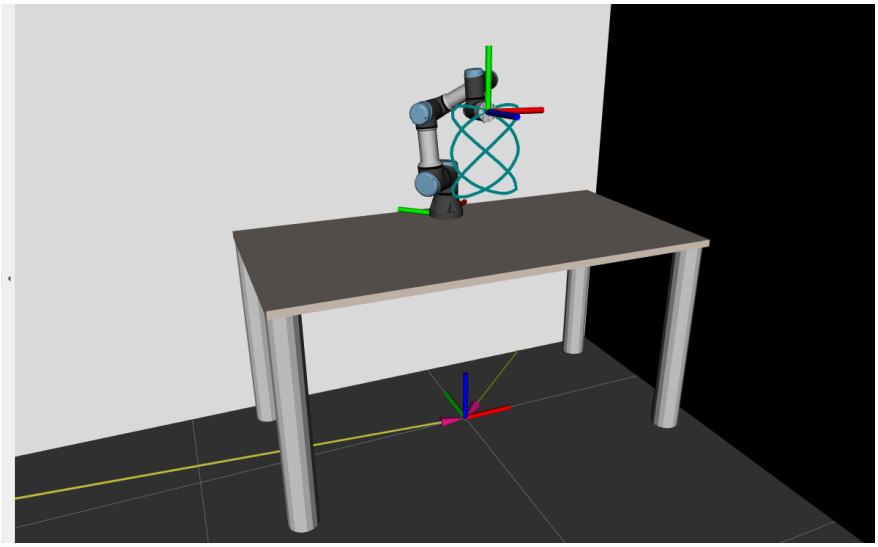

Figure 3.4: Trajectory in RViz

4 Bonus Trajectory Figures

If you chose to create some bonus assignments, briefly describe the trajectory which you are trying to create. How did you go about creating the goal trajectory? Did you do any sort of scripting to procedurally generate your target trajectory, or was it designed by hand? Did you utilize the on/off feature of the LED on the end-effector? What was the most difficult part of this bonus assignment for you, and what did you learn from the trajectory design process? For my bonus assignment, I decided to make a simple smiley face. The idea was to have the robot draw a big circular head, two smaller circular eyes, a short vertical nose, and a curved smile, all as one continuous toolpath that the UR arm could follow. I generated the target trajectory procedurally in MATLAB. Each part of the face was described with basic geometry: a large circle for the head, two smaller circles for the eyes with their own centers and radii, a straight line segment for the nose, and a circular arc for the mouth positioned just below the nose. I built one time vector for the whole motion and used it to stitch these pieces together so that the tool moved continuously from one feature to the next. I did use the LED on the end–effector. The LED is turned on (value 1 in the last column of the CSV) when the robot is actually "drawing" the visible parts of the face: the outer circle, both eyes, the nose, and the smile. When the robot moves between features, the LED is turned off (value 0), so those connecting motions do not show up in the final light painting. The hardest part of this bonus was not the geometry of the smiley itself but getting a trajectory that behaved nicely on the real robot. I had to keep the end–effector speed under about 0.5 m/s and make sure the joint velocities stayed reasonable, while also avoiding kinematic singularities. That meant playing with the timing (total duration, acceleration time in the trapezoidal profile) and checking the plots of end–effector speed, first–order differences in joint angles, and the determinant of the body Jacobian. I went through a few iterations, fixing small issues like sharp changes in speed or parts of the mouth sitting too low, before the path looked smooth in both the plots and RViz.

Figure 4.1: First Order Difference in joint angles. (Similar to Fig. 2.6)

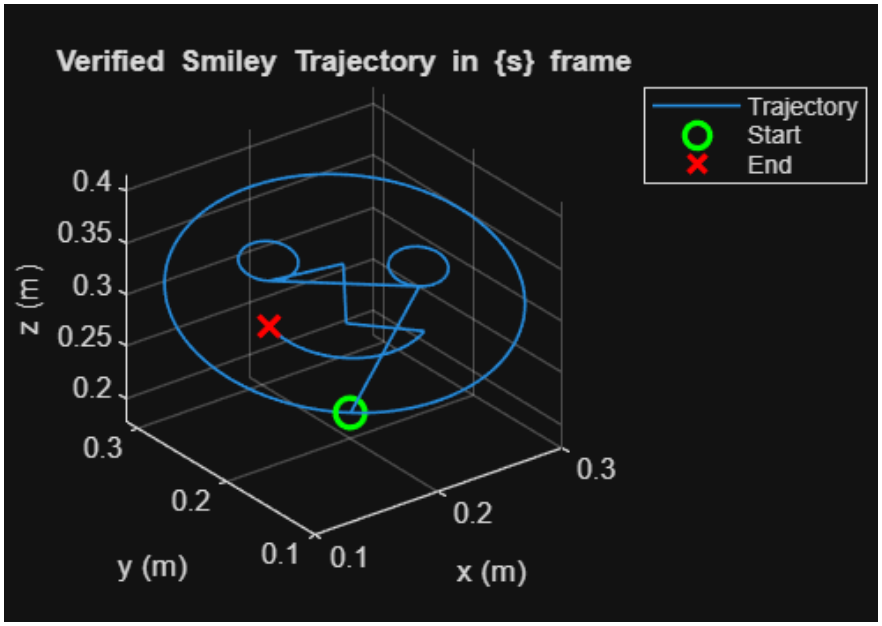

Figure 4.2: Verified Trajectory (using forward kinematics on the output of our inverse kinematics function). (Similar to Fig. 2.7)

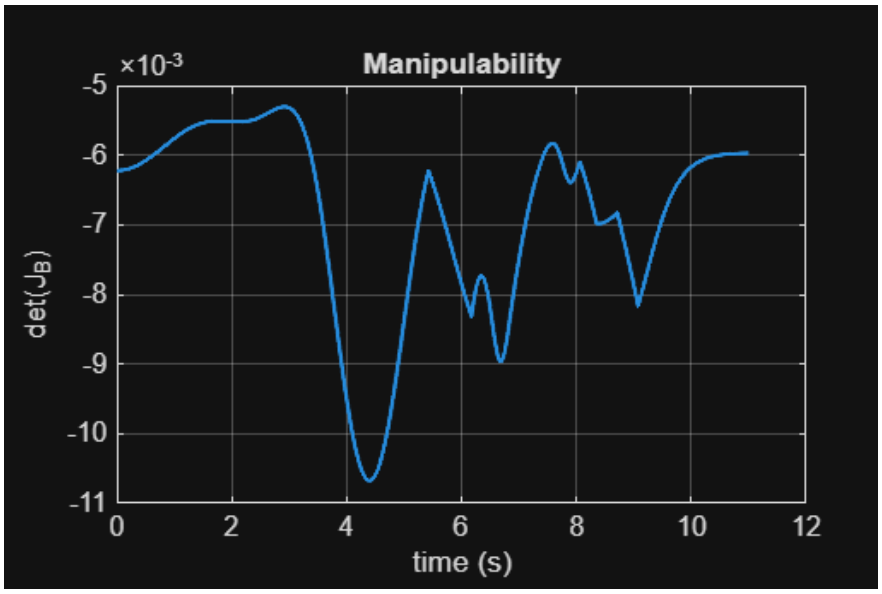

Figure 4.3: Plot of Determinant of Body Jacobian. The plot should stay away from zero, indicating that there are no kinematic singularities.(Similar to Fig. 2.8)

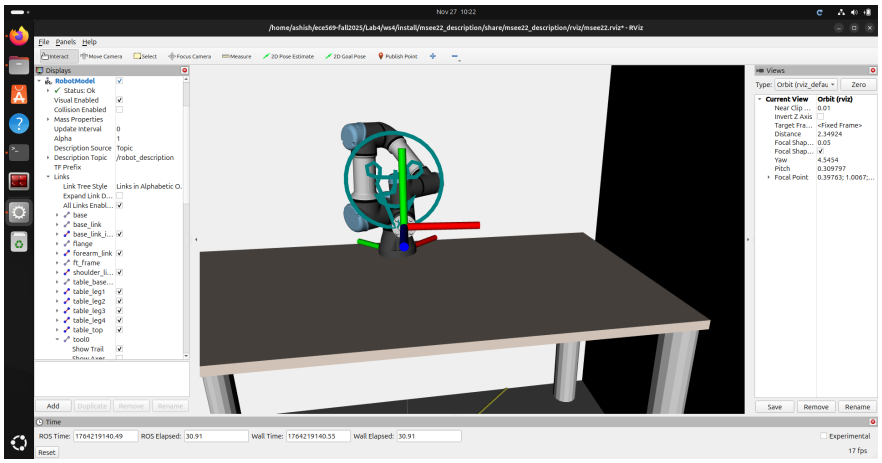

Figure 4.4: If you submitted a bonus CSV trajectory, please take a screenshot of your bonus trajectory RViz, similar to Fig. 3.4.

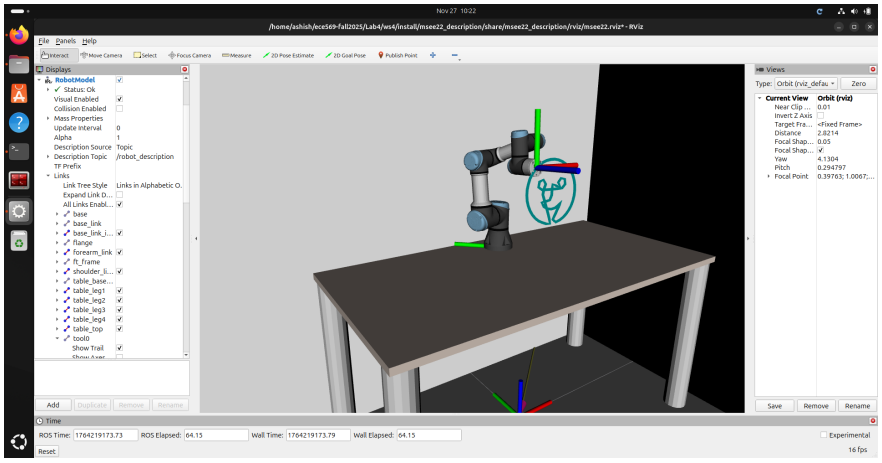

Figure 4.5: If you submitted a bonus CSV trajectory, please take a screenshot of your bonus trajectory RViz, similar to Fig. 3.4.

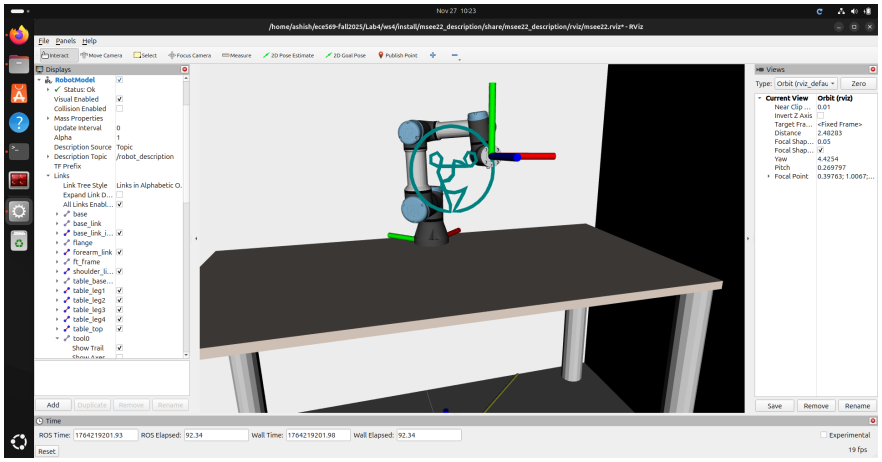

Figure 4.6: If you submitted a bonus CSV trajectory, please take a screenshot of your bonus trajectory RViz, similar to Fig. 3.4.

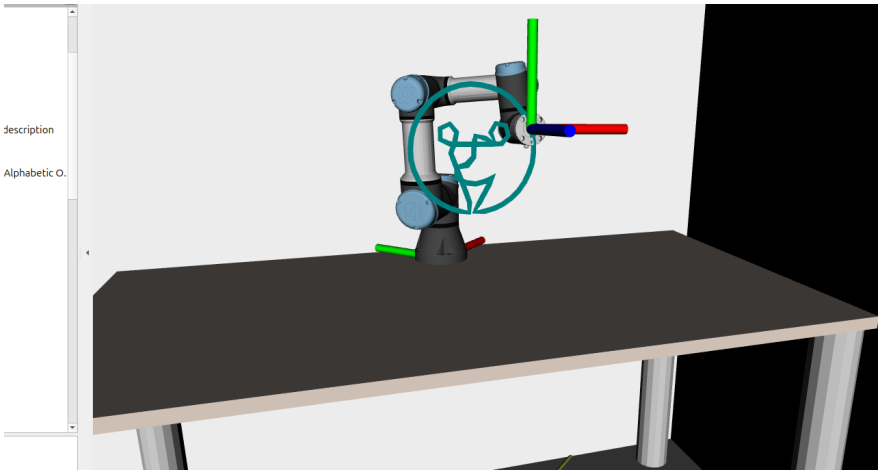

Figure 4.7: If you submitted a bonus CSV trajectory, please take a screenshot of your bonus trajectory RViz, similar to Fig. 3.4.